



CAPSTONE PROJECT

NEWS ARTICLES CLASSIFIER

Group-25

Gnana Venkata Ram Mohan and Swapna M

**Table of Contents**

**PROJECT OBJECTIVE3**

**PROBLEM STATEMENT3**

**PROJECT SETUP 3**

**CODING5**

**INFRASTRUCTURE & DEPLOYMENT8**

**1. OBJECTIVE:**

Design and implement a project that classifies the news feeds into a category using below 3 services.

1. Design and implement a service that collects the news feeds from different sources and store them in Database.
2. Design and implement a service that creates the data frames from data in Database and retrain the model to improve the accuracy
3. Design and implement a service that predicts the news category from the model.

**2. PROBLEM STATEMENT:**

Classify News Articles into categories - With information overload today users are inundated with news articles of all topics, even the ones which may not be relevant to users. Design a system which can classify incoming news articles and appropriately tag the corresponding category. Develop a data pipeline which includes the all the following stages of Machine Learning Project Life Cycle –

1. Data Ingestion

2. Data Preparation

3. Data segregation & Model Training

4. Model Deployment

5. Model Prediction

**3. PROJECT SETUP**

**Stages are elaborated as follows –**

1. **DATA INGESTION**

Project Name - data-ingestion-service

The objective of this project is to collect news feeds and store them in MongoDB.

This data will be used as raw data to retrain the modal in later stages of the project.

We have collected news feeds from two different sources mentioned below.

1) Using rapidAPI URL :

<https://rapidapi.com/newscatcher-api-newscatcher-api-default/api/free-news/>

2) Using newsApi package in Python.

The collected data from these two sources has been translated into below 5 fields while inserting them in MongoDB

• title

• date/ time

• summary

• topic/ category

• source

Below is the data-ingestion-service project data flow diagram:

**Main.py**

This program will trigger two news sources newsAPIService.py and newsrapidAPI.py

**newsAPIService.py**

This program will collect data from NewsApiClient and send them to Kafka topic (news).

**newsrapidAPI.py**

This program will collect data from rapidAPI and send them to Kafka topic (news).

**messageconsumer.py**

This program will consume news feeds from Kafka topic (news) and insert them to MongoDB collection

Db Name : [Capstone](file:///C:\Users\kgvrm\AppData\Local\MongoDBCompass\app-1.28.4\resources\app.asar\src\app\index.html)

Collection Name : news

**Code for Data Ingestion:**

**Main.py**

import newsrapidAPI  
import newsAPIService  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 title=["Finance","Sports","Politics","Religion","Education","Entertainment","Health","Business","cryptocurrency","environment","Crime"]  
 for i in title:  
 print(i)  
 newsrapidAPI.getNews(i)  
 newsAPIService.getNews(i)

**newsAPIService.py**

from newsapi import NewsApiClient  
from kafka import KafkaProducer  
import json  
import time  
  
  
def getNews(title):  
 newsapi = NewsApiClient(api\_key='934545bbdbc14506a2624791a03d6fb9')  
  
 top\_headlines = newsapi.get\_top\_headlines(q=title,  
 category='business',  
 language='en',  
 country='us')  
  
 def json\_serializer(newsDist):  
 return json.dumps(newsDist).encode("utf-8")  
  
 newsDist = {}  
 i=0  
 for i in range(len(top\_headlines['articles'])):  
 newsDist.update(title=top\_headlines['articles'][i]['title'],date=top\_headlines['articles'][i]['publishedAt'],summary=top\_headlines['articles'][i]['content'],category=title,source=top\_headlines['articles'][i]['url'])  
 producer = KafkaProducer(bootstrap\_servers=['localhost:9092'], value\_serializer=json\_serializer)  
 producer.send('news', json.dumps(newsDist))  
 time.sleep(5)  
 print (json.dumps(newsDist))

**newsrapidAPI.py**

import requests  
from kafka import KafkaProducer  
import time  
import json  
  
def getNews(title):  
 url = "https://free-news.p.rapidapi.com/v1/search"  
 querystring = {"q": title, "lang": "en", "page": "1", "page\_size": "25"}  
 headers = {  
 'x-rapidapi-key': "3b1d7a3252mshc8b5142455609abp1c8c1ejsn164eebb8369c",  
 'x-rapidapi-host': "free-news.p.rapidapi.com"  
 }  
  
 response = requests.request("GET", url, headers=headers, params=querystring)  
 response = response.json()  
 print (response)  
 def json\_serializer(newsDist):  
 return json.dumps(newsDist).encode("utf-8")  
 newsDist = {}  
 i=0  
 for i in range(len(response['articles'])):  
 if (response['articles'][i]['topic']=='news' or 'News' or 'NEWS'):  
 category=title  
 else:  
 category=response['articles'][i]['topic']  
  
 newsDist.update(title=response['articles'][i]['title'], date=response['articles'][i]['published\_date'],  
 summary=response['articles'][i]['summary'], category=category,  
 source=response['articles'][i]['link'])  
 producer = KafkaProducer(bootstrap\_servers=['localhost:9092'], value\_serializer=json\_serializer)  
 producer.send('news', json.dumps(newsDist))  
 i=i+1  
 time.sleep(5)  
 print (json.dumps(newsDist))  
  
#getNews('Business')

**messageconsumer.py**

from kafka import KafkaConsumer  
import json  
from pymongo import MongoClient  
  
consumer = KafkaConsumer('news', bootstrap\_servers=['localhost:9092'])  
client = MongoClient("mongodb+srv://Capstone:Capstone@capstone.itrtq.mongodb.net/Capstone?retryWrites=true&w=majority")  
mydb = client["Capstone"]  
mycoll = mydb["news"]  
  
  
for message in consumer:  
 record = json.loads(message.value)  
 news\_record=json.loads(record)  
 mycoll.insert\_one(news\_record)  
 print (news\_record)

**Sample out after data inserted int to MongoDB:**

Graphical user interface, application, table

Description automatically generated

**2. DATA PREPARATION (PRE-PROCESSING), SEGREGATION AND MODEL TRAINING**

Project Name – model-training-service

The objective of this project is to trigger model re-training and deployment on-demand.

Before using the raw data can be used for model training/ retraining it needs to be preprocessed to relevant structure.

• Load the data from “raw\_data” source (MongoDB) into Spark by using relevant connector for PySpark

• Perform data cleaning and preprocessing, followed by segregation to train and test datasets.

1) remove null records

2) RegexTokenizer

3) stopwordsRemover

4) countVectors

•After pre-processing data frame will be split into train data test data in 70/30 ratio

• Perform NaiveBayes model re-training .

• Then Model will be saved to a folder “model” for future reuse.

**ml\_utils.py**

This program Will cérate the NaiveBayes model and train it with data in MongoDB.

It will also provide functions to test sample predictions and retrain the model again.

**main.py**

This program will provide a FastAPI UI support for ml\_utils.py functionality .

**Code:**

**ml\_utils.py**

from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, lower, regexp\_replace, split, udf, trim, concat  
import pyspark.ml.feature  
from pyspark.ml.feature import Tokenizer,StopWordsRemover,CountVectorizer,IDF,RegexTokenizer  
from pyspark.ml.feature import StringIndexer  
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, NaiveBayes  
from pyspark.ml import Pipeline  
from pyspark.ml.evaluation import MulticlassClassificationEvaluator  
from pyspark.sql.types import StringType  
from pyspark import SparkContext  
import pickle  
import numpy as np  
import joblib  
from pyspark.ml import Pipeline, PipelineModel  
  
  
my\_spark1 = SparkSession \  
 .builder \  
 .appName("myApp1") \  
 .config("spark.mongodb.input.uri",  
 "mongodb+srv://Capstone:Capstone@capstone.itrtq.mongodb.net/Capstone.news") \  
 .config("spark.mongodb.output.uri",  
 "mongodb+srv://Capstone:Capstone@capstone.itrtq.mongodb.net/Capstone.news") \  
 .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector\_2.12:3.0.0") \  
 .getOrCreate()  
  
  
def load\_model():  
 df = my\_spark1.read.format("mongo").load()  
 df = df.drop('\_id')  
 df = df.drop('date')  
 df = df.drop('source')  
 df = df.select((concat(col("summary"), col("title")).alias("description")), col("category").alias("category"))  
 df = df.na.drop()  
 # df.groupBy("category").count().show()  
 df.groupBy("category") \  
 .count() \  
 .orderBy(col("count").desc()) \  
 .show()  
 # regular expression tokenizer  
 regexTokenizer = RegexTokenizer(inputCol="description", outputCol="words", pattern="\\W")  
 # stop words  
 add\_stopwords = ["http", "https", "amp", "rt", "t", "c", "the"]  
 stopwordsRemover = StopWordsRemover(inputCol="words", outputCol="filtered").setStopWords(add\_stopwords)  
 # bag of words count  
 countVectors = CountVectorizer(inputCol="filtered", outputCol="features", vocabSize=10000, minDF=5)  
  
 from pyspark.ml import Pipeline  
 from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler  
 label\_stringIdx = StringIndexer(inputCol="category", outputCol="label")  
  
  
  
 (trainingData, testData) = df.randomSplit([0.7, 0.3], seed = 100)  
 print("Training Dataset Count: " + str(trainingData.count()))  
 print("Test Dataset Count: " + str(testData.count()))  
  
 #Naive Bayes  
 nb = NaiveBayes(smoothing=1)  
 pipeline = Pipeline(stages=[regexTokenizer, stopwordsRemover, countVectors, label\_stringIdx,nb])  
 global model  
 model = pipeline.fit(trainingData)  
 predictions = model.transform(testData)  
 predictions.filter(predictions['prediction'] == 0) \  
 .select("description", "category", "probability", "label", "prediction") \  
 .orderBy("probability", ascending=False) \  
 .show(n=10, truncate=30)  
 dataFrame =predictions.dropDuplicates(["category"])  
 dataFrame.show(20)  
  
 evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")  
 evaluator.evaluate(predictions)  
 accuracy = evaluator.evaluate(predictions)  
 print("Model accuracy", accuracy)  
 model.write().overwrite().save("../models")  
  
  
  
# function to predict the flower using the model  
def predict(query\_data):  
 classes = {0: 'Business', 1: 'Politics', 2: 'Entertainment', 3: 'Health', 4: 'Sports', 5: 'Finance',  
 6: 'Religion', 7: 'Education', 8: 'cryptocurrency', 9: 'environment'}  
 x = list(query\_data.dict().values())  
 var1 = x[0]  
 var2 = x[1]  
 news = " ".join([var1, var2])  
 ex1 = my\_spark1.createDataFrame([(news,StringType())],["description"])  
 ex1=ex1.drop('\_2')#----------------------  
 category = model.transform(ex1)#-------------------  
 result=category.select("prediction").collect()[0][0]  
 return classes[result]  
  
#load\_model()

**main.py**

import uvicorn  
from fastapi import FastAPI  
from pydantic import BaseModel  
from ml\_utils import load\_model, predict  
from typing import List  
import datetime;  
  
# defining the main app  
app = FastAPI(title="News Predictor", docs\_url="/")  
  
# calling the load\_model during startup.  
# this will train the model and keep it loaded for prediction.  
app.add\_event\_handler("startup", load\_model)  
  
# class which is expected in the payload  
class QueryIn(BaseModel):  
 title: str  
 summary: str  
  
  
# class which is returned in the response  
class QueryOut(BaseModel):  
 category: str  
  
# class which is expected in the payload while re-training  
class FeedbackIn(BaseModel):  
 title:str= "Retrain the Model with the train data exist in MongoDB"  
  
# Route definitions  
@app.get("/ping")  
# Healthcheck route to ensure that the API is up and running  
def ping():  
 return {"ping": "pong"}  
  
  
@app.post("/predict\_category", response\_model=QueryOut, status\_code=200)  
# Route to do the prediction using the ML model defined.  
# Payload: QueryIn containing the parameters  
# Response: QueryOut containing the category predicted (200)  
def predict\_category(query\_data: QueryIn):  
 #input=list(query\_data.dict().values())  
 #print (input[0])  
 #print (type(input))  
 output = {"category": predict(query\_data)}  
 return output  
  
@app.post("/feedback\_loop", status\_code=200)  
# Route to further train the model based on user input in form of feedback loop  
# Payload: FeedbackIn containing the parameters and correct flower class  
# Response: Dict with detail confirming success (200)  
def feedback\_loop(data:FeedbackIn):  
 load\_model()  
 return {"detail": "Feedback loop successful"}  
  
  
# Main function to start the app when main.py is called  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Uvicorn is used to run the server and listen for incoming API requests on 0.0.0.0:8888  
 uvicorn.run("main:app", host="0.0.0.0", port=8888, reload=True)

**Sample Output:**

Graphical user interface, text, application, email

Description automatically generated

**3.** **MODEL PREDICTION**

Project Name – model-prediction-service

The main objective of this project is to load he pretrained NaiveBayes Model from the folder “models” and take input from HTML based UI and display the news category –

**ml\_utils.py:**

This program will load the saved NaiveBayes Model from “models” folder and predict news category for the input came from HTML UI.

**main.py:**

This program will do the UI support and input data formatting for ml\_utils.py using Flask ,beautifulSoup and requests packages in python.

**Capstone.HTML**

This is a simple UI that takes the new article URL as input and send it to main.py and displays the result the category after prediction

**Project architecture diagram**

A picture containing diagram

Description automatically generated

**4. INFRASTRUCTURE & DEPLOYMENT** –

Additional guidelines

1. Docker images may be created for all the 3 projects. These images can then be used for deployment as containers.

2. Deployment can be orchestrated by using docker-compose (optional)

3. Flask APIs (wherever required) should use Gunicorn/ Bjoern/ CherryPy as a WSGI server.

4. Use PEP guidelines for python code standard.

**ML Tools**

Following tools to be used for project setup –

1. PyCharm as Python IDE

2. Virtual Environment – use venv or virtualenvwrapper to setup separate environments for all projects described.

3. MySQL/ MongoDB as datastore.

4. PySpark for stream processing.

5. POSTMAN for testing Flask APIs

. 6. Apache Zookeeper + Kafka for message queue/ streams.

7. Tensorboard for monitoring the progress of model retraining.

8. MLFlow for model versioning + hyper-parameters versioning.

9. Python cookiecutter templates may be used for setting up the project